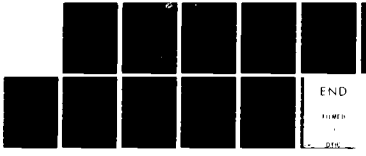
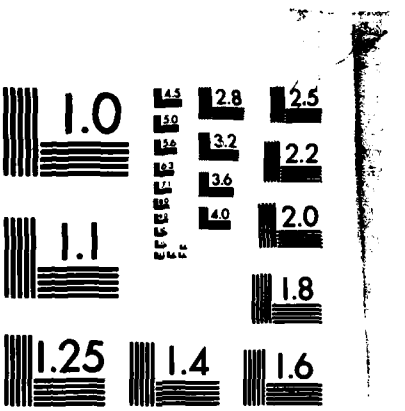
AD-A131 479 DESIGN OF A SYSTEM THAT UNDERSTANDS INFORMAL SPECIFICATIONS(U) DELAWARE UNIV NEWARK DEPT OF COMPUTER AND INFORMATION SCIENCES R M WEISCHEDEL ET AL. APR 83 1/1

UNCLASSIFIED AFOSR-TR-83-0675 AFOSR-80-0190 F/G 9/2 NL





DESIGN OF A SYSTEM THAT UNDERSTANDS INFORMAL SPECIFICATIONS[1]

Ralph M. Weischedel
Daniel L. Chester

Computer & Information Sciences
University of Delaware
Newark, DE 19711

April, 1983

[1]RESEARCH SPONSORED BY THE AIR FORCE OFFICE OF SCIENTIFIC RESEARCH, AIR FORCE SYSTEM COMMAND, USAF, UNDER GRANT NUMBER AFOSR-80-0190. THE UNITED STATES GOVERNMENT IS AUTHORIZED TO REPRODUCE AND DISTRIBUTE REPRINTS FOR GOVERNMENTAL PUPOSES NOTWITHSTANDING ANY COPYRIGHT NOTATION HEREIN.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER **AFOSR-TR- 83-0675**	2. GOVT ACCESSION NO. AD-A131 479
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Design of a System That Understands Informal Specifications	5. TYPE OF REPORT & PERIOD COVERED interim technical report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ralph M. Weischedel Daniel L. Chester	8. CONTRACT OR GRANT NUMBER(s) Grant No. AFOSR-80-0190
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer & Information Sciences University of Delaware Newark, DE 19711	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB Washington, DC 20332	12. REPORT DATE April, 1983 13. NUMBER OF PAGES 16
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

DTIC COPY INSPECTED

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

formal specifications, English specifications, modules, natural language processing, abstract data types, logic programming, Horn clauses

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper describes a system for understanding English definitions of software modules and generating formal specifications. The design of the system itself is emphasized, particularly

- the choice of a target specification language
- selection of a parsing strategy, and
- treatment of semantic problems, such as understanding spatial metaphor and interpreting known words in new environments.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DESIGN OF A SYSTEM THAT UNDERSTANDS INFORMAL SPECIFICATIONS[1]

Ralph M. Weischedel
Daniel L. Chester

Computer & Information Sciences
University of Delaware
Newark, DE 19711

28 April 1983

ABSTRACT

This paper describes a system for understanding English definitions of software modules and generating formal specifications. The design of the system itself is emphasized, particularly

- the choice of a target specification language,
- selection of a parsing strategy, and
- treatment of semantic problems, such as understanding spatial metaphor and interpreting known words in new environments.

1. INTRODUCTION

A cornerstone of the design of large software systems is the definition of their modules based on the information-hiding principle [Parnas 72]. Given that principle, one should define a module interface without revealing the module's (internal) implementation.

One could define a module interface using a formal language such as SPECIAL [Roubine 76] or AFFIRM [Guttag 78, Musser 79] or using natural language. Formal specifications of modules offer many advantages for design of large software systems, including lack of ambiguity, precision, attention to detail, mechanical processing, and appropriateness for both proof techniques and transformations. Nevertheless, few would argue that they are either easy to create or easy to understand once created.

[Balzer 78] argues that some aspects of informality are actually desirable in specifying software modules. Futhermore, [Hobbs 77] cites several aspects of natural langauge semantics which are preferable to existing formal languages.

This paper investigates an artificial intelligence approach to combining the

---

[1]RESEARCH SPONSORED BY THE AIR FORCE OFFICE OF SCIENTIFIC RESEARCH, AIR FORCE SYSTEM COMMAND, USAF, UNDER GRANT NUMBER AFOSR-80-0190. THE UNITED STATES GOVERNMENT IS AUTHORIZED TO REPRODUCE AND DISTRIBUTE REPRINTS FOR GOVERNMENTAL PURPOSES NOTWITHSTANDING ANY COPYRIGHT NOTATION HEREIN.

advantages of both formal and natural languages. The long-term goal is a system which could take as input an English definition of a module, and generate an equivalent formal specification. In addition, the system should generate an English paraphrase of its understanding of the input, so that the user[2] may easily check the system's understanding.

The remainder of the paper describes the design decisions made in implementing a prototype to understand English texts defining data structures. Section 2 enumerates some of the reasons we feel are most important for using natural language. Section 3 defines the target specification language and the motivation in selecting it. Section 4 relates our experience in using a parser for texts defining data structures. Section 5 deals with semantic issues such as interpreting spatial metaphors and selecting precise translations of vague English terms. Related work and our conclusions are presented in sections 6 and 7.

2. THE MOTIVATION FOR NATURAL LANGUAGE

It is our contention that a key to human understanding is having a model of the concept to be understood. For complex concepts, the model may not be very precise, but does provide a framework within which detail can be organized. Present formal languages are weak at expressing such models, unless they are stated in natural language as comments or other accompanying explanations.

If this contention is true, we should see numerous examples of English linguistic expressions used to convey a view of a software module. In examining simple definitions of data structures, we indeed find that. For example, spatial metaphors are prevalent in data structure descriptions, even though the spatial metaphor has little to do with the actual physical realization of any implementation. For instance, one speaks of *the ends of a list*, *the top of a stack*, *running down a list*, *following a pointer*, etc. Such expressions are simple metaphors in that they provide a spatial view which is independent of software or hardware existing in three dimensions.

Analogy also supports this contention, since analogy describes one concept in terms of another, and since it is intended at a general, rather than detailed, level. For instance, one finds definitions such as *A stack is a queue in which all insertions and deletions occur at one end* in [Horowitz 76]. Similarly, in the English definition of the design of the kernel of a secure operating system [Ford 78], one finds the argument to a system call defined as *the sticky bit of UNIX*[3].

Yet another instance is the use of vague terms to connote certain impressions. In defining a queue in terms of an ordered list, [Horowitz 76] defines the enqueue operation as *adding an element i to the rear of a queue*, even though no operation of

---

[2] The user of this AI system is specifying a new system. He/she is expert in the application area.

[3] UNIX is a trademark of Bell Laboratories.

adding has been defined for ordered lists. Rather a notion of adding something to a sequence is presumed, as well as the ability to generalize the notion to the new data structure.

If our contention is correct, one should see concepts introduced specifically to summarize information. For instance, in the English description of a provably secure operating system [Ford 78], one finds the statement, *A SEID is returned as the result of all new object creations (K_build_segment, ...)*. This introduces the concept of interface operations that create new objects; it uses that concept to summarize the fact that six operations are somewhat similar in purpose and return the same class of value.

One could argue that this evidence is not compelling, that all that is needed is richer formal language semantics. For instance, one could try to have a rigorous, precise definition of analogy incorporated in formal languages. Yet, the prevalence and value of mnemonics for procedure names, arguments, constants, etc. is strongly suggestive that no matter what the formal language, a critical part of the understanding of formal specifications will depend on providing an individual with intuitive concepts as a framework for organizing detail. If purely formal languages were adequate, then mathematics after all of these centuries would be stated exclusively in formalisms.

Based on our contention that a key to human understanding is having a model of the concept to be understood, natural langauge has many desirable qualities for specification. Other motivations are presented in [Balzer 78] and [Hobbs 77].

3. SELECTION OF TARGET LANGAUGE AND KNOWLEDGE REPRESENTATION LANGUAGE

The target formal specification language selected is Horn clauses. A Horn clause has the form

C IF A1 & A2 & ... & An

where n>=0, C is an atomic formula, and each Ai is an atomic formula[4]. A Horn clause therefore is a restricted form of formula in first order logic. Since all variables are free, all variables are implicitly universally quantified.

A set of Horn clauses stating axioms about the operations at the interface of a software module can serve as a specification of the module. A primary reason for selecting Horn clauses as the target langauge is that they are also highly appropriate as a knowledge representation language for artificial intelligence. Furthermore, Horn clause theorem provers [Chester 80] are one approach to modelling reasoning. Consequently, once a user's English specification has been understood, adding the set of axioms to the knowledge base of the system offers two potentials:

---

[4] An atomic formula is a predicate applied to terms. A term is a constant, a variable, or a function applied to terms.

- the system can grow in competence,
- the user may define new entities in terms of modules defined earlier, and
- the user may run the theorem prover to test the correctness of the specification or to prove properties of the specification.

As an example, consider the Horn clauses below. All predicates (including equalities and inequalites) are expressed in a LISP-like prefix notation; variables are prefixed by an underscore; and (__X . __Y) is an abbreviation for a term (f __X __Y) expressing the result of adding __X at the front of __Y.

1. (>= (LENGTH __S) 0) IF (TUPLE __S)
2. (TUPLE __S) IF (NULL __S)
3. (TUPLE (__X . __Y)) IF (TUPLE __Y) & (CONSISTENT-TYPE (__X . __Y) __T)

They correspond to the following facts about tuples respectively:

1. A tuple has a nonnegative length.
2. The null tuple is a tuple.
3. A form whose first element is __X is a tuple if the rest of the form is and if the form has a consistent type __T.

From the first Horn clause, we notice that facts that could be expressed with existentially quantified variables will involve Skolem functions rather than existential quantification.

Formal specification languages are a topic of research. It is possible that Gist [Balzer 82] or Tecton [Kapur 82] will lead to languages more appropriate to our task. Both are attempting to incorporate semantics closer to that of natural language as used in informal specifications. If successful, the understanding task may be somewhat simplified, though traditional problems in natural language such as ambiguity, vagueness, reference resolution, metaphor, etc. will still exist.

4. PARSING AND SEMANTIC INTERPRETATION

Given the selected target language, there are several alternatives for the style of grammar and of semantic interpretation. For instance, rather than accounting for all words in a text, one could use a strategy of processing only phrases that are relevant to a domain-dependent schema of stereotypical concepts [Schank 80]. This strategy seems inappropriate for the application of module specification, since in defining a non-stereotypical structure any phrase skipped may be critical to the definition.

Another decision is whether to select a general-purpose grammar having broad coverage of English and then to add special phrases, constructions, and lexical items peculiar to the application. The advantage of this of course is the potential of using someone else's grammar rather than creating one tailored to the domain. Another alternative is to build one specific to the application; semantic grammars [Burton 76] exemplify this. A semantic grammar has nonterminals specific to semantic classes of the

domain, such as data structure phrases, rather than syntactic constituents, such as noun phrases. Because of tight coupling between a phrase and a semantic category, senseless interpretations can be rapidly discarded.

We chose RUS [Bobrow 78], a broad coverage grammar of English which performs semantic interpretaion incrementally. For each constitutent y found and proposed as part of a constituent x, the grammar calls the semantic component to extend the semantic representation of x based on finding y. If y's interpretation is inconsistent with semantic constraints on adding it to x, then the parser abandons this parse. When the grammar proposes that a constituent is complete, the semantic component receives a message. Either it returns a semantic interpretation for x or it vetoes the proposed parse of x.

The semantic component therefore must be prepared to build structures incrementally, rather than waiting till the end of a constituent x before applying semantic constraints on it. The constraints encoded in our semantic interpreter are organized in *case frames*, a very common style of encoding selection restrictions. Each phrase has a head lexical item, e.g. verbs for typical clauses, common nouns for typical noun phrases, and prepositions for prepostional phrases. Any lexical item that can serve as a head may have several case frames associated with it, one frame per sense of the head word. A frame is a list of possible phrase slots that may be associated with a word sense; for each slot a semantic constraint is associated, limiting the kind of entity that can fill the slot. For instance, *delete* has three slots. The logical subject must be a program or person; the logical object must be a data entity; a prepositional phrase whose head is *from* must have an object which is a data structure. That is, one sense of *delete* is that a person or a program may delete a data entity from a data structure. In addition, a slot may be marked as optional or mandatory. Each case frame also includes a structure-building operation, stating what logical expression is to be built for this form.

Processing within the semantic component falls into three cases.

- As soon as the proposed head of a phrase x is found, all case frames are retrieved as possible word senses.
- As a phrase y is proposed to fill a given slot in x, the semantic constraint of that slot is tested on y, potentially eliminating some of the case frames from further consideration. If none remain, y cannot be added to x.
- When the parser proposes that x is complete, the semantic component eliminates any case frame with unfilled mandatory slots and also builds the semantic representation for any remaining case frame.

In using this approach three aspects of our experience are interesting. First, the major modification to the RUS grammar was to allow mathematical notation, so that one can use it freely within English. Thus, a text such as the following can be parsed by the modified grammar[5] (The sentences have been numbered for expository purposes.)

---

[5] This is a modified version of a definition given on pages 41-42 of [Horowitz 76].

1. We say that an ordered list is empty or it can be written as (A[1], A[2], ...,A[N]) where the A[I] are atoms from some set S.
2. There are a variety of operations that are performed on these lists.
3. These operations include the following.
4. Find the length N of the list.
5. Retrieve the ith element, 1<=I<=N.
6. Store a new value at the ith position, 1<=I<=N.
7. Insert a new element at postion I, 1<=I<=N+1 causing elements numbered I,I+1, ...,N to become numbered I+1, I+2, ..., N+1.
8. Delete the element at postion I, 1<=I<=N causing elements numbered I+1, ..., N to become numbered I, I+1, ..., N−1.

The modifications were easy to make, for the patterns with which the mathematical expressions occur fit naturally into the grammar of English.

Second, becuase of a broad-coverage grammar, adding new texts requires proportionally little time on the syntactic aspects. Some new dictionary entries are required, and very infrequently a new construction must be added. Therefore, one can concentrate on the semantic issues, which dominate the effort in extending the system.

Third, RUS's calls to the semantic component eliminates many senseless interpretations. For the text above, the first interpretation found by the parser/semantic component was the correct one in all but one sentence. Furthermore, five of the sentences yielded only one interpretation; the other three yielded only two. As the semantic component is expanded to broader and broader domains, the case frame constraints will be somewhat less effective. For instance, one could expect that there would be 5 interpretations for the first sentence, only one for the next four, and two for the last three in broader environments. Nevertheless, this is radically less than the number of interpretations if no selection restrictions were applied during parsing.

Based on these observations, we feel the time is ripe to adopt broad coverage grammars of English which interact with semantic components to prune senseless parses. The alternative of writing one's own grammar requires substantial time, which could be devoted to other purposes.

5. ADDITIONAL SEMANTIC PROBLEMS

Semantic interpretation, definite reference resolution, and quantifier scope decisions, are well-known semantic problems of natural langauge understanding. Yet, even after a system has generated a semantic representation R where such decisions have been made, there may still be a need for further transformation and understanding of the input to generate a representation S for the underlying application system. There are at least three reasons for this.

First, consider spatial metaphor. Understanding spatial metaphor seems to require computing some concrete interpretation S for the metaphor; however, understanding the metaphor concretely may be attempted after computing a semantic representation R that represents the spatial metaphor formally but without full understanding. Generating an English paraphrase of the system-generated formal specification to allow the user to check the system's undersanding is likely to be both easier and more understandable to the user if the user's terminology is employed. By having an intermediate level of understanding such as R, and generating English output from it, one may not have to recreate the metaphor, for the terms in R use it as a primitive.

Second, the needs of the underlying application system may dictate transformations that are neither essential to understanding the English text nor linguisticly motivated. In a data base environment, transformations of the semantic representation may yield a retrieval request that is computationally less demanding [King 80]. To promote portability, EUFID [Templeton 83] and TQA [Damerau 81] are interfaces that have a separate component for transformations specific to the data base. In software specification, mapping of the semantic representation R may yield a form S which is more amenable for proving theorems about the specification or for rewriting it into some standard form.

The following example, derived from a definition of stacks on page 77 of [Horowitz 76] illustrates these first two reasons. *A stack is an ordered list in which all insertions and deletions occur at one end called the top.* A theorem prover for abstract data types would normally assume that the end of the stack in question is referred to by a notation such as A[1] if A is the name of the stack, rather than understanding the spatial metaphor "one end".

Third, it may be convenient to design the transformation process in two phases, where the output of both phases is a semantic representation. In our system, we have chosen to map certain paraphrases into a common form via a two step process. The forms "ith element" and "element i" each generate the same term as a result of semantic interpretation. However, the semantic interpreter generates another term for "element at position i" due to the extra lexical items "at" and "position". Obviously, all three expressions correspond to one concept. The system must recognize that the two terms generated by the semantic interpreter are paraphrases and map them into one form.

In our system, the semantic representation R is in the form of Horn clauses. All semantic interpretation, quantifier scope decisions, and reference resolution has been performed prior to this second translation phase which is performed by the *mapping component*. Input to the mapping component for the text defining ordered lists is given in the appendix.

The rules of the mapping component are all encoded as Horn clauses. The antecedent atomic formulas of our rules specify either

1. the structural change to be made in the collection of formulas or

2. conditions which are not structural in nature but which must be true if the mapping is to apply.

We will use the notation (MAPPING-RULE (a1 ... am) __x (c1 ... ck) __y) to mean that the atomic formulas a1 ... am must be present in the list __x of atomic formulas; the list __x of formulas is assumed to be implicitly conjoined. The variable __y will be bound to the result of replacing the formulas a1, ..., am in __x with the formulas c1, ..., ck. There is a map between two lists, __x and __y, of atomic formulas if (MAP __x __y) is true.

The two examples given earlier are detailed next. For expository purposes the rules given in this section have been simplified.

Consider the following example: *A stack is an ordered list in which all insertions and deletions occur at one end called the top. ADD(I,S) adds item I to stack S.* In this environment spatial metaphors tend to be more frozen than creative. To understand "one end", we assume the following rules:

1. For a sequence __D, we may map "__E is an end of __D" to "__E is the first sequence element of __D".

2. An ordered list is a sequence.

Facts (1) and (2) are encoded as Horn clauses below.

```
(MAP __X __Y)  IF  (MAPPING-RULE ((END __E __D)) __X
                                 ((SEQUENCE-ELEMENT __E 1 __D)) __Y) &
                   (SEQUENCE __D)

(SEQUENCE __D)  IF  (ORDERED-LIST __D)
```

The system knows how to map the notion of "end of a sequence", and it knows that ordered lists are sequences. Since the first sentence is discussing the end of an ordered list, the two rules above are sufficient to map "end" into the appropriate concrete semantic representation. The power and generality of this approach is that

- a chain of reasoning may show how to view some entity __D as a sequence (and therfore the rules show how to interpret "end of __D"), and

- other mapping rules may state how to interpret spatial metaphors unrelated to "end" or to sequences.

We propose that the same mechanism can deal with certain vague, extended uses of words, such as *add* in the previous example. In stating that ADD(I,S) adds item I to stack S, *add* cannot be predefined, since its meaning is being defined for stacks. Nevertheless, it is reasonable to assume that there is a general relation between "add" and related concepts such as uniting, including, or, in the data structure environments, inserting. Consequently, we propose the following fact in addition to the two above:

- For a sequence __S, we may map "add __I to __S" to "insert __I at some position __X of __S".

It may be stated formally as

(MAP __W __Z) IF (MAPPING-RULE ((ADD __I __S)) __W ((INSERT __I __S __X)) __Z)
& (SEQUENCE __S)

Notice that __X will be unbound. However, the Horn clauses generated for the first sentence (*A stack is an ordered list in which all insertions and deletions occur at one end called the top*) will imply that __X is the position corresponding to the end called top. Therefore, the vague, extended use of "add" can be understood using the inference mechanism of the mapping component. Other rules may state how to interpret an extended use of *add* by relating it to views other than sequences.

Another problem involves mapping the forms "ith element", "element i", and "element at position i" into the same representation. Assume that the semantic interpreter generates for each of the first two the list of formulas ((ELEMENT __X) (IDENTIFIED-BY __X __Y)). The Horn clause for that mapping is as follows:

(MAP __W __Z) IF (TOPIC __T) & (SEQUENCE __T) &
(MAPPING-RULE ((ELEMENT __X) (IDENTIFIED-BY __X __Y)) __W
((SEQUENCE-ELEMENT __X __Y __T)) __Z)

Note that this rule assumes that in context some sequence __T has been identified as the topic; the rule identifies that the element __X is the __Yth member of the sequence __T. For the phrase "element at position i", assume the semantic interpreter generates the list of formulas ((ELEMENT __X) (AT __X (POSITION __P)) (IDENTIFIED-BY __P __Y)). The mapping rule for it is similar to the one above.

(MAP __W __Z) IF (TOPIC __T) & (SEQUENCE __T) &
(MAPPING-RULE
((ELEMENT __X) (AT __X (POSITION __P)) (IDENTIFIED-BY __P __Y))
__W ((SEQUENCE-ELEMENT __X __Y __T)) __Z)

This second rule must be tried before the prior one.

The mapper halts when no more rules can be applied.

6. RELATED WORK

A number of applied AI systems have been developed to support automating software construction [Balzer 78, Green 76, Biermann 80, Gomez 82]. Of these, our effort is the only one that has focussed on the linguistic issues in the mapping problem. It is also distinguished by our design decisions regarding the target langauge and parsing/semantic interpretation. The systems in [Green 76, Biermann 80, Gomez 82] were designed for generating algorithms from English input. In algorithm generation, efficiency of the algorithm generated is of critical concern. This problem is not critical in module specification, since the specification forms a contract stating what programs implementing the specification must do.

Viewing spatial metaphors in terms of a scale was proposed in [Hobbs 77]. Our model is somewhat more general in that the inference process

- permits specific constraints for each metaphor, not just the one view of a scale, and
- accounts for other mapping problems in addition to spatial metaphor.

A very similar approach to mapping has been proposed in [Mark 80]. Instead of using Horn clauses as the formalism for mapping, they encode their rules in KL-ONE [Brachman 78]. The concern in [Mark 80] is inferring the appropriate service to perform in response to a user request, rather than demonstrating means of interpreting spatial metaphors or of finding contextually dependent paraphrases.

The value of generating a paraphrase for a formal specification has been discussed in [Swartout 82]. Language generation is a very active area of research; an overview of the sate of the art is provided in [Mann 81]. No generation component has been included in our prototype system.

7. CONCLUSIONS

The design of a system to generate formal specifications from natural language definitions is a long term research goal. The availability of broad-coverage grammars [Bobrow 78, Robinson 82, Sager 81] that use selection restrictions while parsing to eliminate anomolous parses is an important step toward that. There are five broad areas for future work:

- formal specification languages with richer semantics so that the level of the target language is closer to that of natural languages,
- development of more flexible, forgiving natural language interfaces [Weischedel 83] that have partial understanding even of poorly formed input,
- extension of the technology to broad areas of specification,
- development of high quality English generation components both for creating a paraphrase of the formal specification and for generating questions to clarify ambiguous or vague aspects of the English definitions, and
- further development of the mapping phase.

There are several reasons why one may want such a mapping phase even after a semantic representation for an utterance has been computed. The advantage of using Horn clauses (or any other deduction mechanism) in this mapping phase is the ability to include nonstructural conditions. This means that the mapping rules may be based on reasoning about context.

There are three areas for further development of the mapping phase:

- generating mapping rules based on additional texts,

- investigating use of the mapping component in reference resolution, and
- developing an indexing technique to run the mapper in a forward chaining mode.

APPENDIX

We include here the actual Horn clauses that serve as the output of the semantic component and as the input to the mapping component. The English that generated the Horn clauses is provided for reference in italics; it is not supplied as input to the mapping component. Ampersands have been inserted for expository purposes. For the first sentence, there is no easy way to convert the disjunction to a Horn clause. Therefore, we generate an extended notation allowing disjunction for that case.

We say that an ordered list is empty or it can be written as (A[1],...,A[N]) where the A[I] are atoms from some set S.

```
((OR (((EMPTY A23) IF (LIST A23) & (ORDER NIL A23)))
     (((EQUIV (A0031 A23) (TUPLE (SUBSCRIPT A 1) ELLIPSIS (SUBSCRIPT A N)))
       IF  (LIST A23) & (ORDER NIL A23))
      ((NOTATION NIL A23 (A0031 A23)) IF (LIST A23) & (ORDER NIL A23))
      ((SET (A0032 A23)) IF (EQUIV A71 (SUBSCRIPT A I)) & (LIST A23)
                            & (ORDER NIL A23))
      ((IDENTIFIED-BY NIL (A0032 A23) S)
       IF (EQUIV A71 (SUBSCRIPT A I)) & (LIST A23) & (ORDER NIL A23))
      ((MEMBERS-OF A71 (A0032 A23))
       IF (EQUIV A71 (SUBSCRIPT A I)) & (LIST A23) & (ORDER NIL A23)))))
```

There are a variety of operations that are performed on these lists.

```
(((VARIETY (A0033 A23))
  IF (OPERATION A29) & (PERFORM NIL A29 A23) & (LIST A23) & (ORDER NIL A23))
 ((MEMBERS-OF A29 (A0033 A23))
  IF (OPERATION A29) & (PERFORM NIL A29 A23) & (LIST A23) & (ORDER NIL A23)))
```

These operations include the following.

```
(((INCLUDE A16 A340) IF (FOLLOW A340) &
          (EQUIV A16 (SETOF A0034
                            (AND (OPERATION A0034) (PERFORM NIL A0034 A23))))
          & (LIST A23) & (ORDER NIL A23)))
```

Find the length, N, of the list.

```
(((EQUIV (A0037 A23) N) IF (LIST A23) & (ORDER NIL A23))
 ((LENGTH (A0038 A23) A23) IF (LIST A23) & (ORDER NIL A23))
 (((EQUIV (A0038 A23) (A0037 A23))) IF (LIST A23) & (ORDER NIL A23))
 ((FOLLOW (FIND NIL (A0038 A23))) IF (LIST A23) & (ORDER NIL A23)))
```

Retrieve the ith element, 1<=I<=N.

```
(((LE 1 I) IF (ELEMENT A22) & (IDENTIFIED-BY NIL A22 I))
 ((LE I N) IF (ELEMENT A22) & (IDENTIFIED-BY NIL A22 I))
 ((FOLLOW (RETRIEVE-FROM NIL A22 NIL)) IF (ELEMENT A22) &
```

```
                                        (IDENTIFIED-BY NIL A22 I)))
```

Store a new value into the ith position, 1<=I<=N.

```
(((LE 1 I) IF (POSITION A33) & (IDENTIFIED-BY NIL A33 I) &
              (VALUE A15) & (NEW A15))
 ((LE I N) IF (POSITION A33) & (IDENTIFIED-BY NIL A33 I) &
              (VALUE A15) & (NEW A15))
 ((FOLLOW (STORE NIL A15 (INTO A33)))
     IF (POSITION A33) & (IDENTIFIED-BY NIL A33 I) & (VALUE A15) & (NEW A15)))
```

Insert a new element at position I, 1<=I<=N+1 causing elements numbered I,I+1,...,N to become numbered I+1,I+2,...,N+1.

```
(((POSITION (A0062 A18)) IF (ELEMENT A18) & (NEW A18))
 ((IDENTIFIED-BY NIL (A0062 A18) I) IF (ELEMENT A18) & (NEW A18))
 ((LE 1 I) IF (ELEMENT A18) & (NEW A18))
 ((LE I (PLUS N 1)) IF (ELEMENT A18) & (NEW A18))
 ((FOLLOW (INSERT NIL A18 NIL (AT (A0062 A18))))
  IF (ELEMENT A18) & (NEW A18))
 ((ITEM-OF (A0063 A54 A18)
           NIL
           (SEQUENCE (PLUS I 1) (PLUS I 2) ELLIPSIS (PLUS N 1)))
  IF (ELEMENT A54) & (ITEM-OF A62 NIL (SEQUENCE I (PLUS I 1) ELLIPSIS N))
    & (IDENTIFIED-BY NIL A54 A62) & (NUMBER A62) & (ELEMENT A18) & (NEW A18))
 ((CAUSE (INSERT NIL A18 NIL (AT (A0062 A18)))
         (COME-ABOUT (AND (IDENTIFIED-BY NIL A54 (A0063 A54 A18))
                          (NUMBER (A0063 A54 A18)))))
  IF (ELEMENT A54) & (ITEM-OF A62 NIL (SEQUENCE I (PLUS I 1) ELLIPSIS N))
    & (IDENTIFIED-BY NIL A54 A62) & (NUMBER A62) & (ELEMENT A18) & (NEW A18)))
```

Delete the element at position I, 1<=I<=N causing elements numbered I+1,...,N to become numbered I,I+1,...,N-1.

```
(((POSITION (A0076 A17)) IF (ELEMENT A17) & (AT A17 A27))
 ((IDENTIFIED-BY NIL (A0076 A17) I) IF (ELEMENT A17) & (AT A17 A27))
 ((LE 1 I) IF (ELEMENT A17) & (AT A17 A27))
 ((LE I N) IF (ELEMENT A17) & (AT A17 A27))
 ((FOLLOW (DELETE NIL A17)) IF (ELEMENT A17) & (AT A17 A27))
 ((ITEM-OF (A0077 A51 A17)
           NIL
           (SEQUENCE I (PLUS I 1) ELLIPSIS (SUB N 1)))
 IF (ELEMENT A51) & (ITEM-OF A59 NIL (SEQUENCE (PLUS I 1) ELLIPSIS N)) &
    (IDENTIFIED-BY NIL A51 A59) & (NUMBER A59) & (ELEMENT A17) & (AT A17 A27))
 ((CAUSE (DELETE NIL A17)
         (COME-ABOUT (AND (IDENTIFIED-BY NIL A51 (A0077 A51 A17))
                          (NUMBER (A0077 A51 A17)))))
 IF (ELEMENT A51) & (ITEM-OF A59 NIL (SEQUENCE (PLUS I 1) ELLIPSIS N))
   (IDENTIFIED-BY NIL A51 A59) & (NUMBER A59) & (ELEMENT A17) & (AT A17 A27)))
```

REFERENCES

[Balzer 78] Robert Balzer, Neil Goldman, and David Wile.
Informality in Program Specification.
IEEE Transactions on Software Engineering SE-4(2), March, 1978.

[Balzer 82] Robert M. Balzer, Neil M. Goldman, and David S. Wile.
Operational Specification as the Basis for Rapid Prototyping.
In *Proceedings of the Second Software Engineering Symposium: Workshop on Rapid Prototyping.* ACM SIGSOFT, April, 1982.

[Biermann 80] Alan W. Biermann and Bruce W. Ballard.
Toward Natural Language Computation.
American Journal of Computational Linguistics 6(2), 1980.

[Bobrow 78] R.J. Bobrow.
The RUS System.
In B.L. Webber, R. Bobrow (editors), *Research in Natural Language Understanding,* . Bolt, Beranek and Newman, Inc., Cambridge, MA, 1978.
BBN Technical Report 3878.

[Brachman 78] Ronald Brachman.
A Structural Paradigm for Representing Knowledge.
Technical Report, Bolt, Beranek, and Newman, Inc., 1978.

[Burton 76] Richard Burton.
Semantic Grammar: An Engineering Techniques for Constructing Natural Language Understanding Systems.
Technical Report, Bolt, Beranek, and Newman, 1976.

[Chester 80] Daniel L. Chester.
HCPRVR: An Interpreter for Logic Programs.
In *Proceedings of the National Conference for Artificial Intelligence,* pages 93-95. American Asociation for Artificial Intelligence, Aug, 1980.

[Damerau 81] Fred J. Damerau.
Operating Statistics for the Transformational Question Answering System.
American Journal of Computational Linguistics 7(1):30-42, 1981.

[Ford 78] Ford Aerospace.
Secure Minicomputer Operating System (KSOS): Computer Program Development Specifications (Type B-5).
Technical Report WDL-TR7932, Ford Aerospace & Communications Corp., 1978.

[Gomez 82] Fernando Gomez.
Towards a Theory of Comprehension of Declarative Contexts.
In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguisitcs,* pages 36-43. Association for Computational Linguistics, June, 1982.

[Green 76] C. Green.
The Design of the PSI Program Synthesis System.
In *Second International Conference on Software Engineering.* IEEE Computer Society, Oct, 1976.

[Guttag 78] John V. Guttag, Ellis Horowitz, and David R. Musser.
Abstract Data Types and Software Validation.
Communications of the ACM 21(12):1048–1063, 1978.

[Hobbs 77] Jerry R. Hobbs.
What the Nature of Natural Language Tells us about how to Make Natural-Langauge-Like Programming More Natural.
In *SIGPLAN Notices*, pages 85–93. SIGPLAN, 1977.

[Horowitz 76] Ellis Horowitz and Sartaj Sahni.
Fundamentals of Data Structures.
Computer Science Press, Woodland Hills, CA, 1976.

[Kapur 82] D. Kapur, A. A. Stepanov, and D. Musser.
Tecton: A Language for Manipulating Generic Objects.
In Yogen Staustrup (editors), *Program Specification: Proceedings of a Workshop, Aarhus, Denmark, August, 1981*, pages 402–414.
Springer-Verlag, 1982.

[King 80] Jonathan J. King.
Intelligent Retrieval Planning.
In *Proceedings of the National Conference for Artificial Intelligence*, pages 243–245. American Asociation for Artificial Intelligence, Aug, 1980.

[Mann 81] William C. Mann, Madeleine Bates, Barbara J. Grosz, David D. McDonald, Kathleen R. McKeown, and William R. Swartout.
Text Generation: The State of the Art and the Literature.
Technical Report ISI/RR-81-9, USC/Information Sciences Institute, 1981.

[Mark 80] William Mark.
Rule-Based Inference in Large Knowledge Bases.
In *Proceedings of the National Conference on Artificial Intelligence.*
American Association for Artificial Intelligence, August, 1980.

[Musser 79] David R. Musser.
Abstract Data Type Specifications in the Affirm System.
In *Proceedings of a Conference on Specifications of Reliable Software.*
IEEE Computer Society, 1979.

[Parnas 72] D. L. Parnas.
On the Criteria to be Used in Decomposing Systems into Modules.
Communications of the ACM 15(12):1053–1058, 1972.

[Robinson 82] Jane Robinson.
DIAGRAM: A Grammar for Dialogues.
Communications of the ACM 25(1):27–47, 1982.

[Roubine 76] Olivier Roubine and Lawrence Robinson.
SPECIAL Reference Manual.
Technical Report CSG-45, Stanford Research Institute, August, 1976.

[Sager 81] N.Sager.
Natural Language Information Processing: A computer Grammar of English and its Applications.
Addison-Wesley Publishing Co., Inc., Reading, MA, 1981.

[Schank 80] R.C. Schank, M. Lebowitz, L. Birnbaum.
An Integrated Understander.
American Journal of Computational Linguistics 6(1):13-30, 1980.

[Swartout 82] William B. Swartout.
GIST English Generator.
In *Proceedings of the National Conference for Artificial Intelligence*, pages 404-409. American Asociation for Artificial Intelligence, Aug, 1982.

[Templeton 83] Marjorie Templeton and John Burger.
Problems in Natural Language Interface fo DBMS with Examples from EUFID.
In *Conference on Natural Language Processing*, pages 3-16. Association for Computational Linguistics, Feb, 1983.

[Weischedel 83] Ralph M. Weischedel.
Introduction to Handling Ill-Formed Input.
In *Proceedings of Conference on Applied Computational Linguistics*, pages 89-92. Association for Computational Linguistics, Feb, 1983.